6.895 Theory of Parallel Systems C. Scott Ananian

Language-level Complex Transactions

Project Proposal

Transactions have long been used as an organizing principle for concurrent operations. Originally formulated as a database concept, Knight, Katz, and Tinker used transactions to speculatively parallelize sequential code [Kni86, TK88, Kat89]. Herlihy and Moss later introduced transactions into the hardware memory system [HM93] in order to efficiently implement Lamport's lock-free synchronization [Lam77]. Pure-software implementations of transactional memory followed [ST95, HLMS03]. Rajwar and Goodman applied hardware transactional memory to speed up traditional lock-based programs [RG02].

As a strictly database concept, transactions have appeared in the E programming language [RCS93] and other specialty database languages. General-purpose languages acquired linguistic support for lock-based transactions and atomic actions in Argus [LS82, WL85, LCJS87, Wei90] and Avalon [Ler88, DHW88]. Transactions have been factored into persistence, undoability, locking, and threads as first-class objects in a Standard ML of New Jersey (SML/NJ) implementation [NW92, HKM+94].

In the database community, a variety of extensions beyond simple single-level transactions exist; for example, chained, concurrent, and overlapping transactions. Moss proposed *nested transactions* as an organizational principle for distributed systems [Mos81]. Nested transactions allow concurrency control within transactions by appropriately serializing subtransactions, and permit parts of a transaction to fail without necessarily aborting the entire transaction.

The goal of this project is to extend previous research by integrating non-blocking multiple-level transactions into a modern object-oriented language, Java. In addition to using these for synchronization (as per Herlihy, Rajwar), non-blocking multiple-level transactions ought to be able to be used to express optimistic and speculative concurrency (Knight). Finally, the "undoability" properties of transactions ought to allow their use for fault-tolerance: a crucial question here is how irreversible external I/O within a transaction ought to be handled.

I propose to design a non-blocking multiple-level transaction system for Java and use it to safely implement speculative parallelization of sequential codes. This is a reasonable goal as I have already written a Java compiler and a single-level transaction system. However, it might be necessary to hook my compiler backend up to cilk in order to get low-overhead threading; this may prove to be a lot of work.

As a fallback, I would concentrate on the design of an efficient multiple-level transaction system and its use to write non-blocking parallel codes. I would evaluate the efficiency of using transactions compared to traditional locks, especially as the size of the transaction increases. Interactions with the OS and with the memory subsystem could be systematically analyzed.

Sean Lie is interested in hardware implementations of transactional memories. As an alternative to this proposal, I might collaborate with him to explore the hardware/software interactions of transactions. Hardware tends to implement short transactions well and efficiently, but has trouble with large and long-lived transactions. Software transactions tend to be slower, but are potentially unbounded in size and length (subject to livelock considerations). One avenue for exploration might be integrating hardware and software implementations such that short/small transactions are fast and large/long transactions complete.

References

[DHW88] David L. Detlefs, Maurice P. Herlihy, and Jeannette M. Wing. Inheritance of synchronization and recovery properties in Avalon/C++. *IEEE Computer*, pages 57–69, December 1988.

[HKM+94] Nicholas Haines, Darrell Kindred, J. Gregory Morrisett, Scott M. Nettles, and Jeannette M. Wing. Composing first-class transactions. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(6):1719–1736, 1994.

[HLMS03] Maurice Herlihy, Victor Luchangco, Mark Moir, and William N. Scherer, III. Software transactional memory for dynamic-sized data structures. In *Proceedings of the 22nd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 92–101. ACM Press, 2003.

[HM93] Maurice Herlihy and J. Eliot B. Moss. Transactional memory: Architectural support for lock-free data structures. In *Proceedings of the 20th Annual International Symposium on Computer Architecture (ISCA)*, pages 289–300. ACM Press, 1993.

[Kat89] Morry Katz. ParaTran: A transparent, transaction-based runtime mechanism for parallel execution of scheme. Technical Report MIT/LCS/TR-454, MIT Laboratory for Computer Science, July 1989.

[Kni86] Tom Knight. An architecture for mostly functional languages. In *Proceedings of the 1986 ACM Conference on LISP and Functional Programming (LFP)*, pages 105–112. ACM Press, 1986.

[Lam77] Leslie Lamport. Concurrent reading and writing. *Communications of the ACM*, 20(11):806–811, November 1977.

[LCJS87] B. Liskov, D. Curtis, P. Johnson, and R. Scheifer. Implementation of Argus. In *Proceedings of the 11th ACM Symposium on Operating Systems Principles (SOSP)*, pages 111–122. ACM Press, 1987.

[Ler88] R. A. Lerner. Reliable servers: Design and implementation in Avalon/C++. In *Proceedings of the 1st International Symposium on Databases for Parallel and Distributed Systems (DPDS)*, pages 13–21. IEEE Computer Society Press, 1988.

[LS82] Barbara Liskov and Robert Scheifler. Guardians and actions: Linguistic support for robust, distributed programs. In *Proceedings of the 9th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 7–19. ACM Press, 1982.

[Mos81] J. Eliot B. Moss. Nested transactions: An approach to reliable distributed computing. Technical Report MIT/LCS/TR-260, MIT Laboratory for Computer Science, April 1981.

[NW92] Scott M. Nettles and Jeannette M. Wing. Persistence + undoability = transactions. In *Proceedings of HICSS-25*, pages 832–843, Washington, DC, 1992. IEEE Computer Society Press. Also CMU-CS-91-173, Carnegie Mellon Univ., Pittsburgh, Pa., August 1991.

[RCS93] Joel E. Richardson, Michael J. Carey, and Daniel T. Schuh. The design of the E programming language. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 15(3):494–534, 1993.

[RG02] Ravi Rajwar and James R. Goodman. Transactional lock-free execution of lock-based programs. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, pages 5–17. ACM Press, 2002.

[ST95] Nir Shavit and Dan Touitou. Software transactional memory. In *Proceedings of the 14th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 204–213. ACM Press, 1995.

[TK88] Pete Tinker and Morry Katz. Parallel execution of sequential scheme with ParaTran. In *Proceedings of the 1988 ACM Conference on LISP and Functional Programming (LFP)*, pages 28–39. ACM Press, 1988.

[Wei90] William E. Weihl. Linguistic support for atomic data types. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(2):178–202, 1990.

[WL85] William Weihl and Barbara Liskov. Implementation of resilient, atomic data types. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7(2):244–269, 1985.